AD-A047 506 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
NETEDI: AN AUGMENTED TRANSITION NETWORK EDITOR.(U)
JUL 77 G D HADDEN N00014-75-C-0612
UNCLASSIFIED T-49 NL

1 OF 1
ADA047506

END
DATE FILMED
1-78
DDC

REPORT T-49 JULY, 1977

mc 12

CSL COORDINATED SCIENCE LABORATORY

NETEDI: AN AUGMENTED TRANSITION FOR AUGMENTED TRANSITION NETWORKS

GEORGE DANIEL HADDEN

NR - 049-364
code - 437

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

(6) NETEDI: AN AUGMENTED TRANSITION NETWORK EDITOR.

by

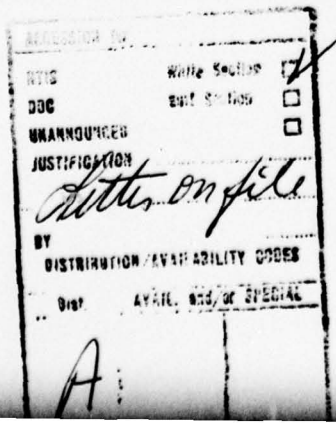
(10) George Daniel/Hadden

This work was supported by the Office of Naval Research under Contract (15) N0014-75-C-0612.

(9) Master's thesis,

(11) Jul 77 (14) T-49

(12) 67p.



092 700

NETEDI: AN AUGMENTED TRANSITION NETWORK EDITOR

BY

GEORGE DANIEL HADDEN

B.S., Purdue University, 1973

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1977

Thesis Adviser: Professor David L. Waltz

Urbana, Illinois

ACKNOWLEDGEMENT

I would like to express my appreciation to Professor David Waltz for his helpful suggestions, encouragement, and patience during my work on this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 TRANSITION NETWORKS

A traditional model for recognizing sentences in certain types of languages has been the "transition network". One of these can be drawn on a piece of paper as a group of circles with labeled directed lines connecting them (see figure 1). The circles are called "states". Generally, one will be designated the "start state" and another the "final state". The lines are called "arcs" and represent ways to get from one state to another if the label is matched by the current input word. Every time an arc is taken, the input advances to the next word.

1.2 A SIMPLE ATN

Now, the transition network in figure 1 can recognize such sentences as "Bob ran." or "Rain falls.", but nothing more complicated. Indeed, any transition network is limited to a small subset of possible English sentences. Why is this? One reason is that sentences can have other sentences embedded in them, and those sentences can have sentences embedded in them, and so on. For instance, consider "The man who owned the dog which bit the thief called the police." This sentence has three levels of embedding.

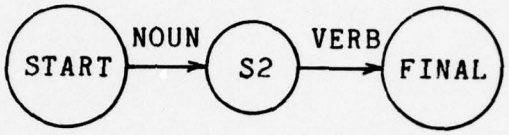
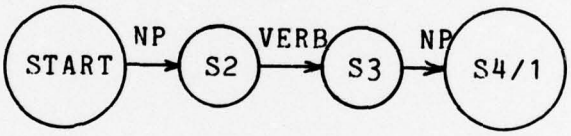


Figure 1 A simple transition network



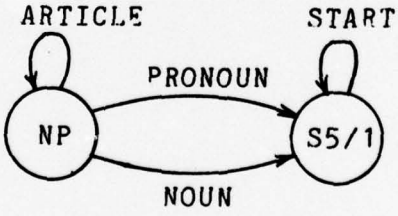


Figure 2 A recursive transition network

If we allow the transition network to be recursive, this sentence (and others) can be recognized. We allow the arcs to have labels which are state names as well as labels which are parts of speech. Then when a state-labeled arc is to be taken, the name of the state pointed to by the arc is put on a push-down list, and we jump to the state named on the arc without advancing the input. This is similar to a subroutine call and is called "pushing to" the state. If we reach a state with a "/1" in it and none of the arcs out of it can be taken, then we jump to the state named at the top of the push-down list, again without advancing the input. This is called "popping". If, however, the push-down is empty and we are out of words at the input, we say that we recognized the sentence.

1.3 AN EXAMPLE

An example might make this a little more clear. We will use the recursive transition network in figure 2 to recognize the example sentence given above. Please refer to table 1 as you read this example.

Starting in the state START, the first arc (the only arc in this case) we see is labeled "NP" (for noun phrase). This is the name of a state so we put "S2", the state pointed to by that arc, onto the push-down list. Then we jump to state NP without advancing the input, i.e. we push to NP. The first word in the sentence is "the", an article, so the ARTICLE arc is taken. Notice we are in state NP again. No matter though; we look at

the next word: "man". This word is a noun, so we take the arc labeled "NOUN" to S5. Now S5 has a "/1" in it which means we pop if we can not take any of the arcs. We can however, and here is where the recursion comes in.

Table 1 Recognizing the sample sentence

STEP	PRESENT STATE	INPUT WORD	PUSH-DOWN LIST (TOP OF LIST TO LEFT)	NEXT ARC TAKEN
1	START	THE	()	PUSH TO NP
2	NP	THE	(S2)	ARTICLE
3	NP	MAN	(S2)	NOUN
4	S5	WHO	(S2)	PUSH TO START
5	START	WHO	(S5 S2)	PUSH TO NP
6	NP	WHO	(S2 S5 S2)	PRONOUN
7	S5	OWNED	(S2 S5 S2)	POP
8	S2	OWNED	(S5 S2)	VERB
9	S3	THE	(S5 S2)	PUSH TO NP
10	NP	THE	(S4 S5 S2)	ARTICLE
11	NP	DOG	(S4 S5 S2)	NOUN
12	S5	WHICH	(S4 S5 S2)	PUSH TO START
13	START	WHICH	(S5 S4 S5 S2)	PUSH TO NP
14	NP	WHICH	(S2 S5 S4 S5 S2)	PRONOUN
15	S5	BIT	(S2 S5 S4 S5 S2)	POP
16	S2	BIT	(S5 S4 S5 S2)	VERB
17	S3	THE	(S5 S4 S5 S2)	PUSH TO NP
18	NP	THE	(S4 S5 S4 S5 S2)	ARTICLE
19	NP	THIEF	(S4 S5 S4 S5 S2)	NOUN
20	S5	CALLED	(S4 S5 S4 S5 S2)	POP
21	S4	CALLED	(S5 S4 S5 S2)	POP
22	S5	CALLED	(S4 S5 S2)	POP
23	S4	CALLED	(S5 S2)	POP
24	S5	CALLED	(S2)	POP
25	S2	CALLED	()	VERB
26	S3	THE	()	PUSH TO NP
27	NP	THE	(S4)	ARTICLE
28	NP	POLICE	(S4)	NOUN
29	S5	--	(S4)	POP
30	S4	--	()	POP

The arc is labeled "START", the name of the first state we saw. Not to be intimidated, though, S5 goes on the push-down list (which has two states on it now), and we push to the state START. We immediately push to NP (S2 goes on the push-down

list), see the word "who", and take the PRONOUN arc to S5.

The next word, "owned", would not be recognized by taking the START arc (try it!), so we pop. That is, we jump without advancing the input to S2, the state on top of the push-down list. Since "owned" is a verb, we can take the VERB arc to S3, where we push to NP again. The push-down list now looks like this: "(S4 S5 S2)", with the top of the list to the left. We are at step 10 in table 1, now.

"The" and "dog" are recognized by the ARTICLE and NOUN arcs, leaving us in S5. We push to START, then to NP and take the PRONOUN arc for "which". We pop to S2 and take the VERB arc for "bit", then push to NP. From here we recognize "the" and "thief" with the ARTICLE and NOUN arcs (a common construct apparently) and pop to S4. Since there are no arcs out of S4 but there is a "/1", we have to pop again. After three more pops we are in S2 looking at "called" at the input. We take the VERB arc, push to NP, and take the ARTICLE and NOUN arcs for "the police". Popping from S5 leaves us in S4 with an empty push-down list. One more pop and (since the push-down list is empty) we have recognized the sentence.

1.4 AUGMENTED TRANSITION NETWORKS

As you can see, recursion tends to make things difficult to follow -- and this was a very simple network. It should be obvious that even this simple network can recognize a certain type of sentence, no matter how many levels of embedding it has.

There are many other instances of sentences where processing of part of the sentence must be suspended while the same network processes another part. But since we do not want only to recognize sentences as belonging to a language, we require additional power. We can expand the concept of recursive transition networks.

First, we can put conditions on the arcs which have to be satisfied before the arc can be taken. Second, we add structure-building machinery which allows us to save parts of the sentence in registers, to move parts of the sentence around to form its deep-structure, and, in our PLANES system [Waltz, et al, 1976], to form commands to our lower level data base language which correspond to questions asked in English.

This expanded concept is called an "augmented transition network" or "ATN". As a mathematical aside, it has the power of a Turing machine [Minsky, 1967].

1.5 NETEDI

These augmented transition networks can be encoded in list structures which, along with an ATN interpreter, will look at an input sentence, parse it according to the ATN grammar, and produce an output. The program NETEDI makes the job of producing and revising the ATNs much easier by leaving much of the busy work to the computer and by taking advantage of certain redundancies in the ATNs.

2
HISTORICAL PERSPECTIVE

2.1 INTRODUCTION

In this chapter, I will discuss three natural language systems which are relevant to NETEDI. The first of these is ELIZA, written by Joseph Weizenbaum, the second is the Teachable Language Comprehender by M. Ross Quillian, and the third is W. A. Woods's system: the Lunar Sciences Natural Language Information System (the Moon Rocks system).

2.2 ELIZA

ELIZA was written at Massachusetts Institute of Technology in 1965 by Joseph Weizenbaum [Weizenbaum, 1966]. Its best known incarnation is as the program called "DOCTOR" -- a simulated Rogerian therapist -- with an astounding ability to convince people that it is a real person.

2.2.1 ELIZA'S OPERATION

Now let us talk about how ELIZA works. First of all, ELIZA scans the input text from left to right looking for keywords. Associated with some keywords is (among other things) a precedence value. If the precedence value of a keyword encountered by ELIZA is greater than it has encountered before in the current input text, then ELIZA puts that word on the top of a stack called the "keystack", otherwise ELIZA puts it at the

bottom of the keystack. During the scan some words may be replaced by substitutes. For instance, "myself" for "yourself", "I" for "you", and so on. This information is stored with the keyword information in the script. A word can be both a substitute and a keyword.

Before we go on, I should explain about the script. The script is a list whose first element is the opening statement, that is, what ELIZA is to print out when the program is entered. The rest of the elements of the script are of the form:

$$(K = S\ N\ ((D_1)(R_{1,1})(R_{1,2}) \ldots (R_{1,m_1}))$$
$$((D_2)(R_{2,1})(R_{2,2}) \ldots (R_{2,m_2}))$$
$$\ldots$$
$$((D_n)(R_{n,1})(R_{n,2}) \ldots (R_{n,m_n})))$$

K is the keyword itself. The equal sign and the "S" following it are optional and indicate that S is to be substituted for K in the input text as was explained above. N is also optional and represents the precedence value. The rest of the entries are the transformation rules. D_i is the i-th decomposition rule and $R_{i,j}$ is the j-th reassembly rule associated with D_i. If these are left out then only the substitution is performed and the word is not considered a keyword and is not entered on the keystack.

This is how the transformation rules work: After the input text has been scanned and the required substitutions made and the keystack has been created, the first transformation rule for the word on the top of the keystack is examined. If the

decomposition rule matches the input text one of its reassembly rules is used to build a response. Now, what does it mean for a decomposition rule to match the input text? A typical decomposition rule for (say) the keyword "you" might look like this:

(0 I remind you of 0)

Where the "you" has been replaced by "I" and vice versa. ELIZA has a simple pattern matcher which when given this as a pattern will allow the 0's to match any sequence of words -- even the empty sequence. For instance this pattern will match the (now changed) input text:

"you think I remind you of your father."

(the user would have typed, "I think you remind me of my father.") If any positive integer is supplied to the matcher in place of the "0", the matcher will require exactly that many words in that portion of the input text. For instance, "(2 I remind you of 0)" would match the above, but "(1 I remind you of 0)" would not.

If ELIZA manages to match a decomposition rule, then the reassembly rules associated with that decomposition rule are put to use. This is done in a cyclic manner, that is, if rule $R_{i,j}$ was used the last time rule D_i is matched, then rule $R_{i,j+1}$ will be used the next time D_i is matched. How are these reassembly rules used? First, notice that the decomposition rule:

(0 I remind you of 0)

is broken up into six parts: (1) "0", (2) "I", (3) "remind", (4) "you", (5) "of", and (6) "0". A reassembly rule for this might

look like the following:

(How do I remind you of 6)

where the "6" refers to the sixth element of the input. So if the input had been "I think you remind me of my father," ELIZA's response would be "How do I remind you of your father."

2.2.2 THE EDITOR

ELIZA contains an editor, called ED which, like NETEDI, makes no reference to line number, page number, or any cues other than contextual ones. It's chief advantage is that by using it, scripts can be augmented on the fly, so to speak. That is, if while working with ELIZA the programmer notices that a response which is not contained in the script, or even a new keyword along with its transformation rules, would improve the performance of the program, he or she can add it by simply suspending the operation (by typing "EDIT") and typing a few simple commands. Then ELIZA can resume conversation using the new script. Similarly, NETEDI can be used in the midst of a session with an augmented transition network processor by simply typing "(CONVERSE)", adding the new sentence-form, and resuming the sessions with a new ATN.

2.3 THE TEACHABLE LANGUAGE COMPREHENDER

2.3.1 TLC'S SEMANTIC NETWORK

M. Ross Quillian's program, Teachable Language Comprehender (TLC) [Quillian, 1969], uses a semantic net for its memory structure. Its memory can be augmented as the program requires it by a human monitor.

The semantic net used by TLC encodes all factual information as either a "unit" or a "property". A unit roughly corresponds to a concept which we would express as a single word, a noun phrase, a sentence, etc., while a property corresponds to a predicate modifying either units or other properties. A unit is represented in the computer's memory as a list whose first element is a pointer to another unit which represents the superset of that unit. For instance, "person" is the superset of both "child" and "golfer". Any other elements of the list, if they exist, are pointers to properties.

Properties are stored in the form of extended attribute-value pairs. What does this mean? An example of an attribute-value pair is "color-red". "Color" is the attribute and "red" is its value. Quillian extends this concept by allowing attributes to be prepositions and verbs and the value to be the appropriate objects. So "plays-baseball" and "beside-river" are both extended attribute-value pairs. In the memory, these again take the form of lists. The first element points to the attribute; the second, to the value. Any other

elements of the list are pointers to other properties.

Since the values pointed to in properties are in general other units, the semantic net is very richly connected. As an example, consider the word "client". Figure 3 shows the structure pointed to by the dictionary entry for this word.

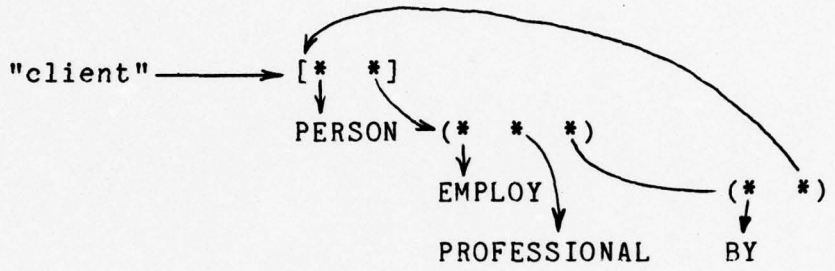


Figure 3 TLC's definition of "client"

Two important things to be aware of are these: First, the words in capital letters represent other units in memory, which in turn contain pointers to still other units, and so on. Second, we see that a subpart of the unit can refer to the unit itself. So TLC's meaning for "client" is "a person who employs a professional."

2.3.2 COMPREHENSION

What does it mean for TLC to "comprehend" text? This means essentially relating textual assertions to items stored in the memory. I should mention here that Quillian considers TLC to be a model of human comprehension. He bases his claim on psychological reaction time tests performed on human subjects. Two problems become immediately apparent here: One, how does TLC choose the relevant parts of the text and the relevant parts of

its memory to relate together. That is, for instance, "Bob shot his mother" could have different interpretations depending on whether Bob is a psychopath or a photographer. Two, what sort of response should TLC make to the text in order to indicate its comprehension of that text?

Let us consider an example of text TLC might receive to see how these problems are solved. If TLC is given the input "lawyer's client", it might respond with this: "Now we are talking about a client who employs a lawyer." Now what has happened here? We see that TLC has identified "lawyer" with "professional" in the definition of "client" given in figure 3. At first glance this might seem an obvious thing to do, but with further consideration we see that TLC has to know that "professional" is a superset of "lawyer", that clients employ professionals, and has had to make the assumption that the lawyer in the text is the professional that the client employs.

Briefly this happens as follows: TLC forms an empty unit for each word in the input text. Along with each of these new units is a list of pointers to possible candidates for the word's meaning. These include all dictionary definitions of the word and all possible anaphoric referents. (For purposes of this discussion, we will ignore the fact that TLC can handle anaphoric referents and just consider word definitions.) Now TLC must supply supersets and properties for these units so they will represent the input.

Using a breadth-first search on the units and properties pointed to from the word definition, TLC looks for intersections. In the "lawyer's client" example it would find an important intersection at the unit "PROFESSIONAL". This unit is the superset of lawyer and the value of the attribute "EMPLOY" in the definition of "client". How is it that TLC knows this intersection is important? "Client" and "lawyer" also have an intersection at "person" -- a comparatively uninteresting one. Part of the reason this is an important intersection is that it is close, i.e. TLC did not have to explore very far around its network to find it. Another reason has to do with the syntactic relationship of the words.

TLC has a minimal concern with syntax. Its syntactic rules are very simple, local ones called "form tests". In this example, the property "EMPLOY" has a form test stored with it which succeeds if the word which is a candidate for its attribute ("LAWYER") has an "'s" following it and the source word ("client") following that. Since this is the case in this example, "lawyer" is identified as the professional in question. It might seem odd here that we call "client" the source word. Why doesn't "lawyer" hold that distinction? After all, it was the first word in the input. The reason is simply this: When the breadth-first search is done, the unit "PROFESSIONAL" is reached first from "LAWYER". A tag is left noting that TLC had come across this unit. Then when the search is done from "CLIENT", TLC sees the tag and only then recognizes that an intersection is taking place.

Now that TLC has made this connection, Quillian says, it can be said to have comprehended the input. The structure shown in figure 4 is built by TLC in comprehending "lawyer's client".

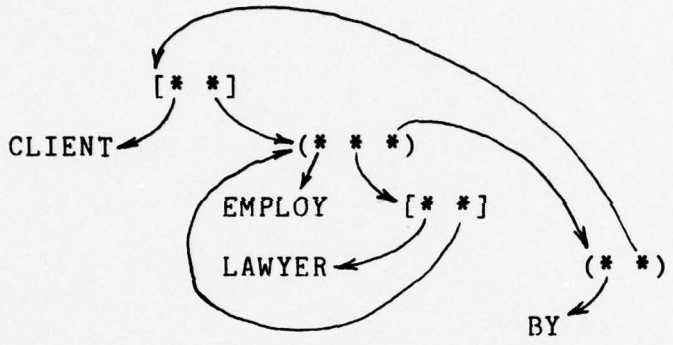


Figure 4 TLC's interpretation of "lawyer's client"

As an aside, I would like to make the suggestion that the use of form tests is a possible explanation for a phenomenon in human conversation. People often will begin a sentence in one tense and finish in another; similarly for number, gender, and so on. These grammatical errors are seldom noticed by either party during the conversation. It might be that the sentence is grammatical taken two consecutive words at a time. If people used something like form tests in generating as well as understanding these sentences, this behavior would be explained.

2.3.3 TEACHING

So what can TLC be taught? First of all, it can be given new form tests. Since a particular form test can be used in many situations, adding one increases understanding greatly. As an example of this effect, notice that the form test used in the example above was associated with "employ" -- a word which did

not even appear in the input text. I suspect that many more sentences use the concept of "employ" than use the word "employ".

The other class of things that TLC can be taught is word definitions. Again an addition of a word definition will increase understanding of sentences which do not use the word explicitly due to the incredible richness of the connections of the network.

The similarity between ELIZA and TLC is that both are meant to start with a relatively small memory. The programmer can then build onto it piece by piece as she or he reacts interactively with the program. TLC is much more sophisticated, however. I think the possibility exists that a TLC-like program could, after a time, teach itself, with less and less interaction from a programmer required as the memory grows.

2.4 THE LUNAR SCIENCES NATURAL LANGUAGE INFORMATION SYSTEM (THE MOON ROCKS SYSTEM)

W. A. Woods deserves a great deal of credit for inventing ATNs [Woods, 1970] and for the excellent job he and his colleagues, Kaplan and Nash-Webber did on the Moon Rocks system [Woods, et al, 1972]. This system contains what is to my knowledge the most complete parser of the English language in existence. Woods claims that seventy-eight percent of the questions asked by geologists unfamiliar with the system were answered.

However, Woods had no special purpose ATN editor. He used a LISP editor which evidently is similar to the one which NETEDI uses as a subroutine [Woods, 1977]. I will explain in section 3 the problems we had when trying to edit ATNs in this way. They are the reasons NETEDI was invented.

2.5 SIMILARITIES TO NETEDI

Of the three systems discussed, ELIZA's editor seems to be most like NETEDI in intent. Both have a simple input language, and both are very interactive. That is, the changes can be made and the new programs used immediately without having to jump back and forth between a text editor or a compiler and LISP (or MADSLIP in the case of ELIZA).

NETEDI is similar to both TLC and ELIZA in that it can start with a small (or nonexistent) system and build it up into a sizable one. An example of this process is in section 4.

The Moon Rocks system and NETEDI are similar mainly in that they both use augmented transition networks as bases for their grammars.

3

THE OPERATION OF NETEDI

3.1 MOTIVATION

When using an Augmented Transition Network, one must occasionally add to it the ability to deal with new types of sentences. Before NETEDI existed, we had three ways of doing this: TECO, SOS (both text editors on the DEC-10 system), and the LISP editor [Finin and Gabriel, 1975]. It happened that most of the new arcs we added to our PLANES system [Waltz, et al, 1976] for such sentence-types were "WRD" arcs. Here is an example of a typical WRD arc:

(WRD WHAT T (TO S:12))

This recognizes the word "what" and jumps to state S:12. (The "T" is a test; any LISP predicate could be inserted in place of it. If it evaluates to "T" the arc is taken.)

An arc similar to this one must be inserted for each word in a new sentence-type. As you can perhaps guess, this gets pretty tiresome. Furthermore, it is not really necessary. Since most of the arcs will be of a similar form, why not let the computer do the busy work? All we should have to tell it is which words we want recognized and in what order and any special actions which should be taken if the word is recognized. The computer can make up a next state, form the arc and insert it into the ATN.

Similar redundancies appear in other types of arcs. NETEDI has been designed to rid the programmer of the necessity of contending with these and also to give him or her a concise, easily understood representation of sentences to be added to the ATN.

3.2 AN EXAMPLE OF NETEDI'S OPERATION

Here is an example of a sentence-form:

```
((WHAT WHICH) &PLANETYPE (:ROOT (NEED REQUIRE))
  &MAINTTYPE IN (MAY (NEXTSTATE END)))
```

This is the kind of input that NETEDI receives and turns into new parts of the ATN. Notice that the sentence-form is in the form of a list. NETEDI scans the list looking at elements one at a time. We shall follow this example through NETEDI assuming we have an ATN as in figure 5. (An actual ATN would be much larger, of course.)

Basically NETEDI is composed of two stages: the tracing stage and the creating stage. During the tracing stage NETEDI tries to match as much of the sentence-form as it can to existing arcs in the ATN. During this stage the ATN is not changed at all -- merely examined.

The first thing NETEDI encounters in the sentence-form is the list "(WHAT WHICH)". Since it is in the tracing stage, it tries to find an identical arc pointing out of the START state. There is one: it points to state S2. This is called a "WRD" arc. NETEDI now looks at state S2 hoping to find an arc to match

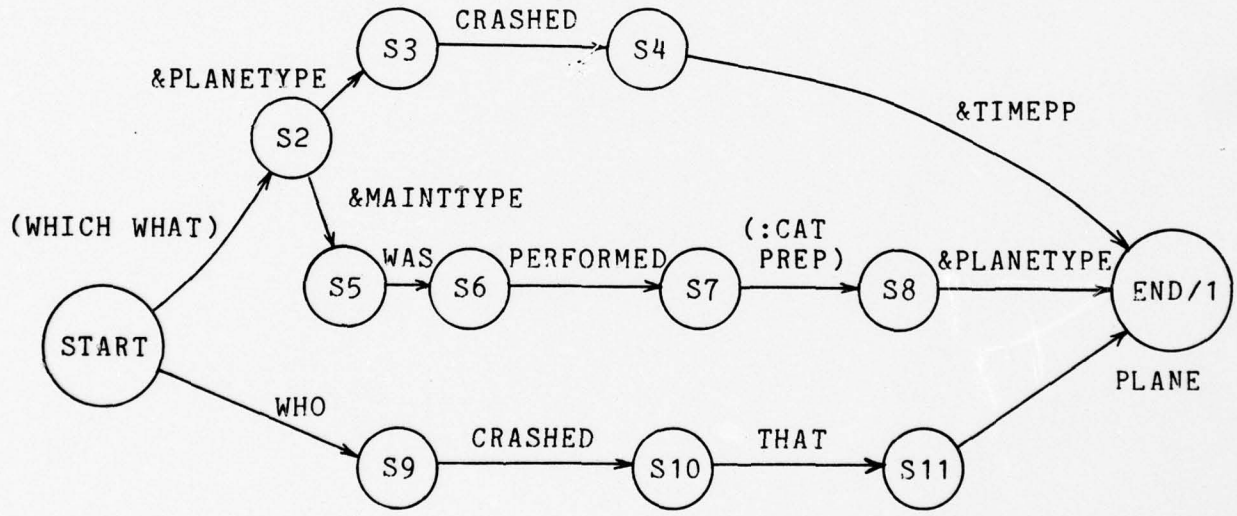


Figure 5 An ATN for NETEDI operation example

"&PLANETYPE" (a "PUSH" arc). Again it is successful: it finds the arc which points to S3. At S3 NETEDI can not find an arc to match the next element in the sentence-form, i.e. "(:ROOT (NEED REQUIRE))", so it shifts into the creating stage.

I should mention here that when NETEDI sees as an element in its sentence-form a single word not starting with "&" (which signifies a PUSH arc), it does not just dive right in and look for an associated WRD arc. Instead, it checks its dictionary for the occurrence of one of two conditions.

First, the ATN interpreter sometimes encounters compound words whose parts it would rather treat separately. For instance, "anything" is really the two words "any" and "thing". If this information is entered in the dictionary for a word and NETEDI encounters that word in one of its sentence forms, it is able to make the necessary substitution. So the sentence-form "(DID ANYTHING CRASH THIS MONTH)" becomes "(DID ANY THING CRASH THIS MONTH)".

The advantage of this tactic is that fewer arcs will be added to the ATN. The reason for this is simple. If a WRD arc for the word "anything" existed in the ATN, it would never be taken since the ATN interpreter begins processing the word by substituting the two words "any" and "thing".

The second condition is this: Sometimes groups of words appear together in a common phrase, e.g. "as well as" or "all right". A mechanism exists in our system to treat these phrases

as one word. This is done by storing appropriate information in the dictionary entry of the first word of the phrase. Each time NETEDI sees a word which might be the beginning of such a phrase, it replaces that phrase with the given single word. For instance the phrase "as well as" gets replaced by the word "aswellas" in the sentence-form and NETEDI tries again. So the sentence-form "(DID A PHANTOM AS WELL AS AN F4 CRASH IN MAY)" is replaced by "(DID A PHANTOM ASWELLAS AN F4 CRASH IN MAY).

The creating stage begins when NETEDI finds that it can not match an element in the sentence-form to an arc in the net. From here on the program must create new states and new arcs in the ATN corresponding to the words in the remaining part of the sentence-form.

"Creating" refers to the new arcs and new states which are generated in order to recognize new sentences. In our example, we are in state S3 and we have to add an arc to recognize (:ROOT (NEED REQUIRE)). (This arc incidentally, will recognize any word whose root form is "need" or "require", e.g. "needed", "requires", etc.)

The first thing we need is a name for a new state -- one which has not yet been used. This is necessary so that the arc will have a place to point to. We can assume that in general arcs begin and end at a state. From figure 5 we can see that the name S12 has not been used yet. So, we create a state with that name and an arc to it (see figure 6).

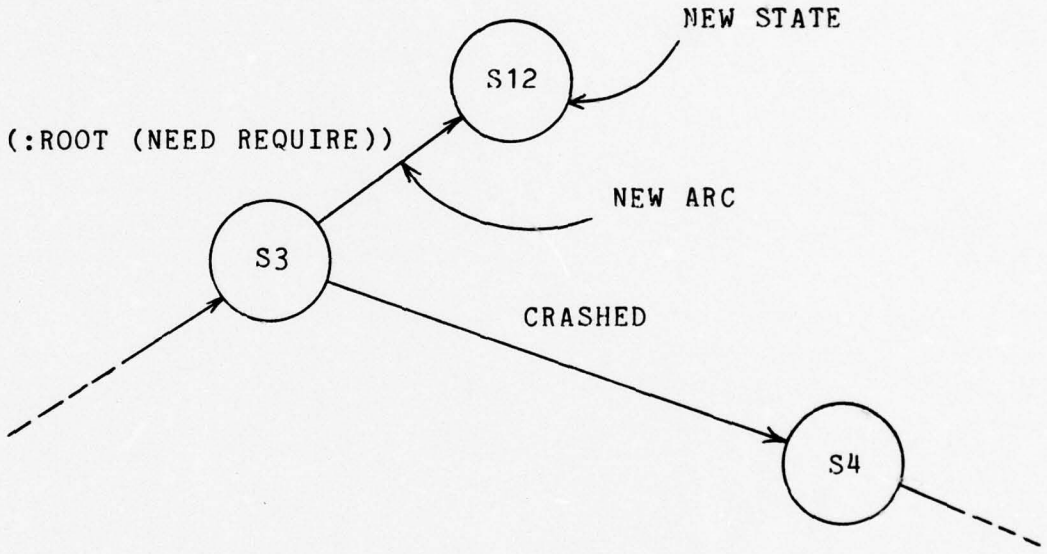


Figure 6 Creating a new state and new arc

This process is continued, adding arcs and states until there are no more words in the sentence-form. When this point is reached, NETEDI can do one of two things.

First of all, the user, after being asked for the sentence-form, has the opportunity to give NETEDI a final arc. This arc is typed verbatim and is handled as if its equivalent had been the last element of the sentence-form. This is advantageous in two respects. Firstly, a naive user, i.e. one with little knowledge of ATNs, will find this feature makes NETEDI easy to use, since she or he can simply type in sentences and similar (possibly prearranged) final arcs. Obviously, this will make for an inefficient ATN, but this is often not one of the naive user's concerns.

The second thing NETEDI can do is to simply stop. In this case the user should have typed as the last element in the sentence-form any one of the "POP" ("POP", "POP2", or "SPOP") arcs or another arc with the next state specified (see section 3.4). We, not being naive users, point the final arc to the already existing "END" state (see figure 7).

In any case, when this point is reached, NETEDI asks the user if he or she would like to continue. If the answer is "yes", the process begins anew. If "no", NETEDI files the completed ATN either from the place where it originally found it or in a new file specified by the user.

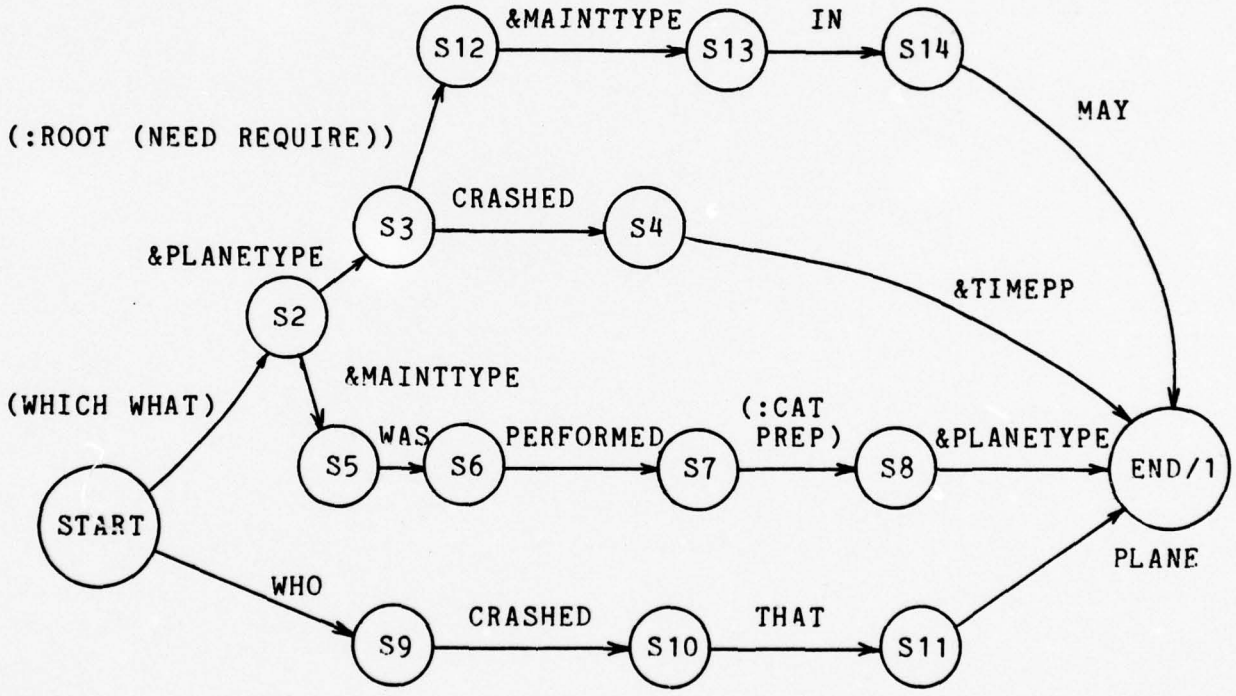


Figure 7 Resulting modified ATN (compare figure 5)

3.3 A PLANES EXAMPLE

An example of NETEDI in action follows. In it, a sample question is presented to PLANES to answer. PLANES is unable to answer it, making it necessary to add the sentence to the ATN. NETEDI is called by the function "CONVERSE". The sentence-form corresponding to the unanswerable sentence (and some others) is typed in along with the associated answer-form, and a few other items. NETEDI thanks the programmer, inserts the sentence-form and answer-form, and notes that fourteen states were added to the ATN.

On the next page the question is asked again. This time the net is able to answer it.

Now suppose we want to add to the network the ability to answer a similar question, e.g., one which matches on some initial segment a sentence already in the ATN. The next page shows such a question. Note that this time only five new states were added. A similar example is on the two pages following these.

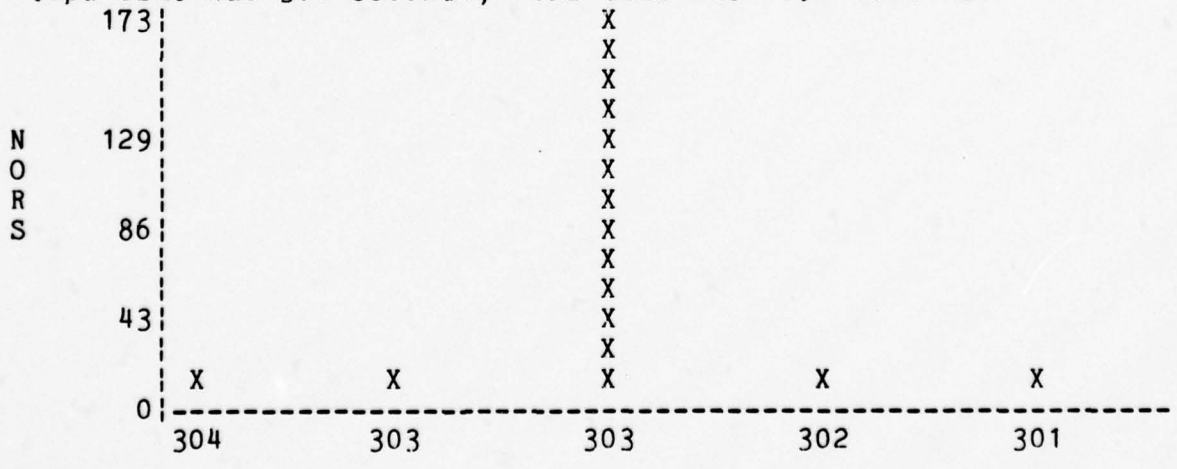
```
; Archive file of 3/12/1976

Please enter your question.....
  >> for each month in /1973 show the nor hours for plane /3


parsing......
 {cpu time was 0.69 seconds, real time was 1.63 seconds.
I could not understand your request Please rephrase it.
```

```
; Archive file of 3/12/1976
T
(CONVERSE)
AUGMENTED TRANSITION NETWORK EDITOR VERSION 1

PLEASE TYPE THE SENTENCE-FORM TO BE ADDED TO THE NETWORK
(FOR EACH MONTH (:CAT PREP) 1973. (:ROOT (GIVE SHOW)) THE NOR
 HOURS FOR PLANE 3)

PLEASE TYPE THE ASSOCIATED ANSWER-FORM
(POP (QUOTE (DATES (PLOT) TIME NIL (NEG NIL) (NOR (PLANE (PRONOUN
     NIL) (TYPE NIL) (BUSER 3)) NIL) (TIME (DATE (MONTH NIL) (DAY
     NIL) (YEAR 73.)) NIL) (NIL NIL))))

WHAT IS THE INITIAL STATE OF THE NETWORK
S0001
WHAT IS THE FILENAME AND PPN OF THE NETWORK TO BE EDITED?
(PLEASE TYPE IN THE FORM:
       ( FILENAME EXT ( PROJECT# PROGRAMMER# )) )
(ATNNEW FOO (1000 423))
loading:   (ATNNEW FOO DSK (1000 423))
;fasloading (ATN FAS DSK (1000 130)) for: DEFATN
       16 s-expressions, 0 functions read.
;fasloading (HATN FAS DSK (1000 130)) for: EDITATN
;loading EDIT.70

;fasloading (EDIT FAS SYS) for: EDIT1
loading EDIT.INI (NIL)
THANK YOU
FOO
14 NEW STATES WERE ADDED TO FOO
nil?
```

```
; Archive file of 3/12/1976

Please enter your question.....
  >> for each month in /1973 show the nor hours for plane /3 .


parsing......
 {cpu time was 1.29 seconds, real time was 3.7 seconds.
I have understood your request as:
     (DATES (PLOT)
          TIME
          NIL
          (NEG NIL)
          (NOR (PLANE (PRONOUN NIL) (TYPE NIL) (BUSER 3.)) NIL)
          (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 73.)) NIL)
          (NIL NIL))

Interpreting.....
 {cpu time was 0.33 seconds, real time was 1.1 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
          '((V O))
          '((V ACTDATE) (V (NOR)))
          '(AND (GEQ (V ACTDATEMON) 1.)
                (LEQ (V ACTDATEMON) 12.)
                (EQU (V ACTDATEYR) 3.)
                (EQU (V BUSER) 3.))
          'NIL)

Evaluating.....
 {cpu time was 3.1 seconds, real time was 4.73 seconds.
     290|                          X
        |                          X
        |                          X
(       |                          X
N    217|                          X
O       |                          X
R       |                          X
)    145|                          X
        |                          X
        |                          X
     72 |                          X
        |                          X
        | X          X             X           X           X
      0 |-----------------------------------------------------
        304        303           303         302         301

                                ACTDATE

          MIN = 0.0          AVERAGE = 58.0        MAX = 290.0
```

```
  (CONVERSE)
AUGMENTED TRANSITION NETWORK EDITOR VERSION 1

PLEASE TYPE THE SENTENCE-FORM TO BE ADDED TO THE NETWORK
(FOR EACH MONTH (:CAT PREP) 1973. (:ROOT (GIVE SHOW)) THE NORS
 HOURS FOR PLANE 3)

PLEASE TYPE THE ASSOCIATED ANSWER-FORM
(POP (QUOTE (DATES (PLOT T) TIME NIL (NEG NIL) (NORS (PLANE
 (PRONOUN NIL) (TYPE NIL) (BUSER 3)) NIL) (TIME (DATE (MONTH
 NIL) (DAY NIL) (YEAR 111)) NIL) (NIL NIL))))

THANK YOU
FOO
5 NEW STATES WERE ADDED TO FOO
NIL
```

```
; Archive file of 3/12/1976

Please enter your question.....
  >> for each month in /1973 show the nors hours for plane /3 .


parsing......
 {cpu time was 0.61 seconds, real time was 1.49 seconds.
I have understood your request as:
   (DATES (PLOT T)
        TIME
        NIL
        (NEG NIL)
        (NORS (PLANE (PRONOUN NIL) (TYPE NIL) (BUSER 3.)) NIL)
        (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 73.)) NIL)
        (NIL NIL))

Interpreting.....
 {cpu time was 0.15 seconds, real time was 0.31 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
         '((V 0))
         '((V ACTDATE) (V NORS))
         '(AND (GEQ (V ACTDATEMON) 1.)
               (LEQ (V ACTDATEMON) 12.)
               (EQU (V ACTDATEYR) 3.)
               (EQU (V BUSER) 3.))
         'NIL)

Evaluating.....
 {cpu time was 3.0 seconds, real time was 6.91 seconds.
```




```
MIN = 0.0          AVERAGE = 34.6          MAX = 173.0
```

```
  (CONVERSE)
AUGMENTED TRANSITION NETWORK EDITOR VERSION 1

PLEASE TYPE THE SENTENCE-FORM TO BE ADDED TO THE NETWORK
(FOR EACH MONTH (:CAT PREP) 1973. (:ROOT (GIVE SHOW)) THE
 NORMU HOURS (:CAT PREP) PLANE 3)

PLEASE TYPE THE ASSOCIATED ANSWER-FORM
(POP (QUOTE (DATES (PLOT T) TIME NIL (NEG NIL) (NORMU (PLANE
 (PRONOUN NIL) (TYPE NIL) (BUSER 3)) NIL) (TIME (DATE (MONTH
 NIL) (DAY NIL) (YEAR 111)) NIL) (NIL NIL))))

THANK YOU
FOO
5 NEW STATES WERE ADDED TO FOO
NIL
```

```
; Archive file of 3/12/1976

Please enter your question.....
  >> for each month in /1973 give the normu hours for plane /3 .


parsing......
 {cpu time was 0.82 seconds, real time was 1.55 seconds.
I have understood your request as:
   (DATES (PLOT T)
          TIME
          NIL
          (NEG NIL)
          (NORMU (PLANE (PRONOUN NIL) (TYPE NIL) (BUSER 3.)) NIL)
          (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 73.)) NIL)
          (NIL NIL))

Interpreting.....
 {cpu time was 0.28 seconds, real time was 0.76 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
           '((V 0))
           '((V ACTDATE) (V NORMUNS))
           '(AND (GEQ (V ACTDATEMON) 1.)
                 (LEQ (V ACTDATEMON) 12.)
                 (EQU (V ACTDATEYR) 3.)
                 (EQU (V BUSER) 3.))
           'NIL)

Evaluating.....
 {cpu time was 3.09 seconds, real time was 4.76 seconds.
     117|                          X
        |                          X
        |                          X
N       |                          X
O       |                          X
R    87 |                          X
M       |                          X
U       |                          X
N    58 |                          X
S       |                          X
        |                          X
    29  |                          X
        |                          X
        | X          X             X            X            X
     0  |-----------------------------------------------------------
        304        303           303          302          301

                                ACTDATE

          MIN = 0.0          AVERAGE = 23.4        MAX = 117.0
```

3.4 SYNTAX OF THE SENTENCE-FORM

3.4.1 THE SENTENCE-FORM

The sentence-form is the most important input to NETEDI. Its form is a list of elements called "proto-arcs" each of which (when NETEDI is finished processing the sentence-form) corresponds to an arc in the ATN. Each one of these arcs may have existed in the ATN before the sentence-form was processed or may have been added during the processing.

3.4.2 THE PROTO-ARC

The proto-arc may have several forms. First of all, it can be a LISP atom not starting with the symbol "&". NETEDI interprets this as an English word, so it forms a WRD arc: "(WRD <atom> T (TO ?STATE))". I should say something here about "?STATE". One of two things can happen to this atom. First, it can be used in a matching function which tries to find out if the arc is already in the ATN. If it is, ?STATE is bound to the next state. Secondly, if the arc is not there, NETEDI replaces ?STATE with a new state name and inserts the arc into the ATN.

On the other hand, if the atom does start with "&", the proto-arc is interpreted as a PUSH arc. The arc is formed as follows: "(PUSH &<atom> T (SETR &<atom> *) (TO ?STATE))". Note that NETEDI adds an instruction to set the register &<atom> to whatever is popped by the subnet. This is almost always useful because it is the only way to remember this information. The

name of this register can be easily changed by adding the characters ">>" and then the desired register name to the end of the proto-arc. So the proto-arc "&NP>>OBJECT" produces the arc "(PUSH &NP T (SETR OBJECT *) (TO ?STATE))".

If the proto-arc is a list of atoms, the first of which does not start with the symbol ":", NETEDI interprets it as corresponding to a WRD arc, where the list is a list of English words. The resulting arc is this: "(WRD (<atom 1> <atom 2> ... <atom n>) T (TO ?STATE))", and it is to be taken by the ATN interpreter if any one of the words in the list is seen at the head of the input string.

The fourth type of proto-arc is a general one which can form any arc at all. This one has the form of a two-element list. The first element must be the name of an arc-type (e.g., CAT, ROOT) preceded by the symbol ":". The second is an argument which becomes the second element in the completed arc. For example the proto-arc "(:ROOT (NEED REQUIRE))" corresponds to the arc "(ROOT (NEED REQUIRE) T (TO ?STATE))".

The only exceptions to this arc format are the POP, POP2, and SPOP arcs. These arcs have no need of a next state specification so "(TO ?STATE)" is not included in them.

The final type of proto-arc is an extension of the above forms. It consists of a list whose first element must be one of the first four types of proto-arcs. The rest of the elements of this proto-arc form a list called the "command list". Each of

its elements must itself be a list. There are five different commands, and, if present, they can be specified in any order.

The first command is called "MARK". The entry in the command list looks like this: "(MARK <atom>)". If NETEDI sees this, it notes the state that the arc formed from the proto-arc is in or is to be added to and associates it with the atom given. This allows the user to label parts of the ATN being built up for later use. One of the features which can take advantage of MARKed states asks where in the ATN to begin adding the sentence-form. If the user types a name which has been associated before with a state in the ATN, the sentence-form will begin in that state. The other feature is explained in the following paragraph.

Another command, "NEXTSTATE", allows the user to point an arc to any state in the ATN. Its format is "(NEXTSTATE <atom>)". Obviously, this can be very useful when used in conjunction with the MARK command. As in the beginning-state question, the user can type either a name which has been used before to MARK a state or an actual state name (if known).

The next command which can be inserted in the command list is "ACTION" (or "ACTIONS"). As before this word is the first element of a list. However in this case the rest of the list can be more than one element long and can be made up of any LISP s-expressions. These are inserted into the arc after the test field (i.e. after the third element). So the proto-arc,

```
((:ROOT BE) (ACTIONS (SETQ BECOUNT (1+ BECOUNT))
                     (PRINT '(FOUND ANOTHER ONE))))
```

becomes the arc,

```
(ROOT BE T (SETQ BECOUNT (1+ BECOUNT))
           (PRINT '(FOUND ANOTHER ONE))
           (TO ?STATE))
```

The test field can be changed to whatever is desired by including "(TEST <s-expression>)" in the command list. The s-expression is put in place of the "T" in the arc. For example, if the proto-arc looks like this: "((:POP 'DONE) (TEST (NULL *)))", the arc would have the form "(POP 'DONE (NULL *))". Note here that there is no "(TO ?STATE)" in the arc. This is also true of POP2 and SPOP arcs since their next state is implied.

The final command is used to change the "(TO ?STATE)" in an arc to "(JUMP ?STATE)". This allows the ATN interpreter to advance to another state without advancing the input. The entry in the command list looks like this: "(JUMP)".

Any of the above commands can be abbreviated in the command list by its first letter.

In section 4 is a scenario in which the features described here are put to use.

4

A Scenario

4.1 The Problem

In this chapter I present an example of NETEDI in use. The task is this: during the course of a session with PLANES [Waltz, et al, 1976], we would like to be able to define one word or phrase as standing for another. For example, "From now on let my favorite planes stand for A7's and F4's." The desired result of typing this sentence would be for the system, whenever it sees the words "my favorite planes", to replace them with the phrase "A7's and F4's".

4.2 The Scenario

This section illustrates the procedure used in building an ATN with NETEDI. I start with a minimal one-state ATN with no arcs. "_" and ">>" are prompting characters--the first for the top level of LISP and the second for NETEDI. Text following these was typed by me to the corresponding program. The "y" or "n" following yes/no questions is also my typing. Indented text was added after the session for explanatory purposes. It will be helpful (possibly necessary) to refer to the completed ATN shown in section 4.3 as you read this scenario.

```
; Archive file of 4/18/77 at 22:29:29
T
_ (DEFATN DEFNET ((DEF1)))
```

DEFNET

This defines the minimal ATN.

_(CONVERSE)

The function call "(CONVERSE)" invokes NETEDI.

Augmented Transition Network Editor Version 2

Please type the sentence-form to be added to the network.

>>(FROM NOW (ON (NEXTSTATE DEF1)))

"NEXTSTATE" gives NETEDI a specific state to which it should go. If it were not there, NETEDI would make one up.

Please type the associated answer-form.

(Type "NIL" if none is desired.)

>> NIL

It is usually easier to use no answer-form, particularly when the commands "NEXTSTATE" and "MARK" are used in the last element of the sentence-form.

In what state should I begin?

>>DEF1

Thank you.

DEFNET

2. new states were added to DEFNET

Would you like to add anything else to the net?

Please type "Y" or "N". y

```
Please type the sentence-form to be added  to the network.
>>((IF WHEN WHENEVER) I (SAY TYPE PRINT) READ IT AS
   ((:POP (QUOTE OK)) (ACTION (MAKEMEAN (GETR &X) (GETR &Y)))
                      (TEST (NULL *))
                      (MARK END)))
```

The "(NULL *)" test is true if there are no more words in the input string.

```
Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
State G0054 marked as END
Thank you.
DEFNET
6. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((IF WHEN WHENEVER) I (SAY TYPE PRINT) (I (MARK M1))
   (MEAN (N END)))
```

Note that we have MARKed a state (M1) for later use. Note also That we have used the initial letter of "NEXTSTATE". This shortcut can be taken with any of the five commands: NEXTSTATE, MARK, ACTION or ACTIONS, TEST, or JUMP.

Please type the associated answer-form.

(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?

Please type "Y" or "N". y

State G0051 marked as M1

Thank you.

DEFNET

1. new states were added to DEFNET

Notice that this sentence-form has the same initial segment as the one before. NETEDI could then use arcs already in the ATN. Consequently only one new state was added. This feature is aimed at a user who is fairly unfamiliar with ATN structure. In the following sentence-form, we see that we can specify a new starting state, hence eliminate much of the typing.

Would you like to add anything else to the net? y

Please type the sentence-form to be added to the network.

>>(((READ SUBSTITUTE) (N END)))

Please type the associated answer-form.

(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?

Please type "Y" or "N". n

In what state should I begin?

>>M1

Here we specify a new starting state (for NETEDI to begin adding arcs -- not for the ATN interpreter), M1, which we MARKed before.

Thank you.

DEFNET

0. new states were added to DEFNET

Would you like to add anything else to the net?

Please type "Y" or "N". y

Please type the sentence-form to be added to the network.

>>((&X (N M1)))

If there is not one already there (which there is not), this generates a PUSH arc which will push to state "&X". It also inserts "(SETR &X *)" into the arc -- a standard procedure with PUSH arcs.

Please type the associated answer-form.

(Type "NIL" if none is desired.)

>> NIL

Start in state G0051 ?

Please type "Y" or "N". y

Thank you.

```
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((&Y (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0051 ?
Please type "Y" or "N". n

In what state should I begin?
>>END
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y

Please type the sentence-form to be added  to the network.
>>((MAKE LET SET) (ALL (M M2)) OCCURRENCES (OF (N M2)))
```

```
Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0054 ?
Please type "Y" or "N". n

In what state should I begin?
>>DEF1
State G0056 marked as M2
Thank you.
DEFNET
3. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y

Please type the sentence-form to be added  to the network.
>>(MEAN THE SAME THING (AS (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL
```

Start in state DEF1 ?

Please type "Y" or "N". n

In what state should I begin?

>>M2

Thank you.

DEFNET

4. new states were added to DEFNET

Would you like to add anything else to the net?

Please type "Y" or "N". y

Please type the sentence-form to be added to the network.

>>(MEAN THE SAME (AS (N END)))

Notice that the sentence-form has the same initial segment as the previous one. The reason the MARK and new starting state features were not used here is that for such a short sentence-form it would have taken more typing.

Please type the associated answer-form.

(Type "NIL" if none is desired.)

>> NIL

Start in state G0056 ?

Please type "Y" or "N". y

Thank you.

DEFNET

```
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((EQUAL (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0056 ?
Please type "Y" or "N". y
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(STAND (FOR (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL
```

```
Start in state G0056 ?
Please type "Y" or "N". y
Thank you.
DEFNET
1. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(((BE MEAN EQUAL) (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0056 ?
Please type "Y" or "N". y
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((&X (N M2)))
```

```
Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0056 ?
Please type "Y" or "N". y
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(DEFINE (TO (M M3) (N M2)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0056 ?
Please type "Y" or "N". n

In what state should I begin?
>>DEF1
```

```
State G0064 marked as M2
Thank you.
DEFNET
1. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y

Please type the sentence-form to be added  to the network.
>>(DEFINE (&X (N M3)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(CONSIDER ((:CAT PREP) (M M4)) (:ROOT (BE MEAN)) THE
   SAME (AS (M M5) (N END)))
```

```
Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
State G0065 marked as M4
Thank you.
DEFNET
5. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((THING (N M5)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". n

In what state should I begin?
```

```
>>M5
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y

Please type the sentence-form to be added  to the network.
>>((:CAT PREP) (:ROOT BE) EQUAL (TO (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0069 ?
Please type "Y" or "N". n

In what state should I begin?
>>M4
Thank you.
DEFNET
2. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y
```

```
Please type the sentence-form to be added  to the network.
>>((:CAT PREP) (:ROOT STAND) (FOR (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0065 ?
Please type "Y" or "N". y
Thank you.
DEFNET
1. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>((:CAT PREP) ((:ROOT (BE MEAN)) (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0065 ?
Please type "Y" or "N". y
```

```
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(REPLACE (WITH (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state G0065 ?
Please type "Y" or "N". n

In what state should I begin?
>>DEF1
Thank you.
DEFNET
1. new states were added to DEFNET

Would you like to add anything else to the net?
Please type "Y" or "N". y

Please type the sentence-form to be added  to the network.
```

```
>>(REPLACE (ALL (M M6)) OCCURRENCES (OF (N M6)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
State G0073 marked as M6
Thank you.
DEFNET
2. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(REPLACE (&X (MARK M7) (N M7)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
State G0073 marked as M7
```

```
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(SUBSTITUTE FOR ((:POP (QUOTE OK))
                   (ACTION (MAKEMEAN (GETR &X) (GETR &Y)))
                   (TEST (NULL *))
                   (MARK XEND)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
State G0077 marked as XEND
Thank you.
DEFNET
2. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
```

```
>>(SUBSTITUTE FOR (&X (N XEND)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(SUBSTITUTE (&Y (N M8)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
Thank you.
DEFNET
```

```
0. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(BY I (MEAN (N END)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL

Start in state DEF1 ?
Please type "Y" or "N". y
Thank you.
DEFNET
2. new states were added to DEFNET

Would you like to add anything else to the net? y

Please type the sentence-form to be added  to the network.
>>(BY (&X (M M9) (N M9)))

Please type the associated answer-form.
(Type "NIL" if none is desired.)

>> NIL
```

```
Start in state DEF1 ?
Please type "Y" or "N". y
State G0078 marked as M9
Thank you.
DEFNET
0. new states were added to DEFNET

Would you like to add anything else to the net? n
Where shall I file DEFNET ?
(Please type in the form:
        (FILENAME EXT DSK (PROJECT# PROGRAMMER#)) )
>> (DEFNET ATN DSK (512. 120.))
```

4.3 The Final ATN

Here is a copy of the ATN generated by the commands shown in the previous section. Also included for completeness are the subnets "&X" and "&Y".

```
(DEFATN DEFNET
      ((DEF1 (WRD FROM T (TO G0047))
             (WRD (IF WHEN WHENEVER) T (TO G0049))
             (WRD (MAKE LET SET) T (TO G0056))
             (WRD DEFINE T (TO G0064))
             (WRD CONSIDER T (TO G0065))
             (WRD REPLACE T (TO G0073))
             (WRD SUBSTITUTE T (TO G0076))
             (WRD BY T (TO G0078)))
       (G0047 (WRD NOW T (TO G0048)))
       (G0048 (WRD ON T (TO DEF1)))
       (G0049 (WRD I T (TO G0050)))
```

```
        (G0050 (WRD (SAY TYPE PRINT) T (TO G0051)))
        (G0051 (WRD READ T (TO G0052))
               (WRD I T (TO G0055))
               (WRD (READ SUBSTITUTE) T (TO G0054))
               (PUSH &X T (SETR &X *) (TO G0051)))
        (G0052 (WRD IT T (TO G0053)))
        (G0053 (WRD AS T (TO G0054)))
        (G0054 (POP 'OK
                    (NULL *)
                    (MAKEMEAN (GETR &X) (GETR &Y)))
               (PUSH &Y T (SETR &Y *) (TO G0054)))
        (G0055 (WRD MEAN T (TO G0054)))
        (G0056 (WRD ALL T (TO G0057))
               (WRD MEAN T (TO G0059))
               (WRD EQUAL T (TO G0054))
               (WRD STAND T (TO G0063))
               (WRD (BE MEAN EQUAL) T (TO G0054))
               (PUSH &X T (SETR &X *) (TO G0056)))
        (G0057 (WRD OCCURRENCES T (TO G0058)))
        (G0058 (WRD OF T (TO G0056)))
        (G0059 (WRD THE T (TO G0060)))
        (G0060 (WRD SAME T (TO G0061)))
        (G0061 (WRD THING T (TO G0062)) (WRD AS T (TO G0054)))
        (G0062 (WRD AS T (TO G0054)))
        (G0063 (WRD FOR T (TO G0054)))
        (G0064 (WRD TO T (TO G0056))
               (PUSH &X T (SETR &X *) (TO G0064)))
        (G0065 (CAT PREP T (TO G0066)))
        (G0066 (ROOT (BE MEAN) T (TO G0067))
               (ROOT BE T (TO G0070))
               (ROOT STAND T (TO G0072))
               (ROOT (BE MEAN) T (TO G0054)))
        (G0067 (WRD THE T (TO G0068)))
        (G0068 (WRD SAME T (TO G0069)))
        (G0069 (WRD AS T (TO G0054)) (WRD THING T (TO G0069)))
        (G0070 (WRD EQUAL T (TO G0071)))
        (G0071 (WRD TO T (TO G0054)))
        (G0072 (WRD FOR T (TO G0054)))
        (G0073 (WRD WITH T (TO G0054))
               (WRD ALL T (TO G0074))
               (PUSH &X T (SETR &X *) (TO G0073)))
        (G0074 (WRD OCCURRENCES T (TO G0075)))
        (G0075 (WRD OF T (TO G0073)))
        (G0076 (WRD FOR T (TO G0077))
               (PUSH &Y T (SETR &Y *) (TO G0040)))
        (G0077 (POP 'OK
                    (NULL *)
                    (MAKEMEAN (GETR &X) (GETR &Y)))
               (PUSH &X T (SETR &X *) (TO G0077)))
        (G0078 (WRD I T (TO G0079))
               (PUSH &X T (SETR &X *) (TO G0078)))
        (G0079 (WRD MEAN T (TO G0054)))))

(DEFATN X
      ((&X (TST FOO T (SETR FOO (LIST *)) (TO G0080)))
```

```
        (G0080 (POP (NCONC (GETR &X 1.) (GETR FOO))))))

(DEFATN Y
      ((&Y (TST FOO T (SETR FOO (LIST *)) (TO G0082)))
       (G0082 (POP (NCONC (GETR &Y 1.) (GETR FOO))))))
```

LITERATURE CITED

[Finin and Gabriel, 1975] Finin, T. W. and Gabriel, R. P.; "The LISP Editor"; Working Paper 1, Advanced Automation Group, Coordinated Science Laboratory, University of Illinois; February, 1975.

[Minsky, 1967] Minsky, M. L.; Computation: Finite and Infinite Machines; Prentice-Hall; Englewood Cliffs, N. J.; 1967.

[Quillian, 1969] Quillian, M. R.; "The Teachable Language Comprehender: A Simulation Program and Theory of Language"; Communications of the ACM; August, 1969.

[Waltz, et al, 1976] Waltz, D. L., et al; "The PLANES System: Natural Language Access to a Large Data Base"; Coordinated Science Laboratory Technical Report T-34; November, 1976.

[Weizenbaum,1966] Weizenbaum, J.; "Eliza -- A Computer Program for the Study of Natural Language Communication Between Man and Machine"; Communications of the ACM; January, 1966.

[Woods, 1970] Woods, W. A.; "Transition Network Grammars for Natural Language Analysis"; Communications of the ACM; October, 1970.

[Woods, et al, 1972] Woods, W. A., Kaplan, R. M., and Nash-Webber, B.; "The Lunar Sciences Natural Language System: Final Report"; Bolt, Beranek, and Newman, Inc., Report No. 2378; Cambridge, Massachusetts; 1972.

[Woods, 1977] Woods, W. A.; personal communication; March, 1977.